Developing a Backend Application using .NET and MSSQL using Visual Studio Code

# Step by step guide for creating a .NET class library

using Visual Studio Code

## Prerequisites

* Visual Studio Code with the C# extension installed.
* If you have the C# Dev Kit extension installed, uninstall or disable it.
* The [.NET 6 SDK](https://dotnet.microsoft.com/download/dotnet/8.0) or NET 8 SDK

A class library defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 8, it can be called by any application that targets .NET 8. This tutorial shows how to target .NET 8.

## Create a solution

Start by creating a blank solution to put the class library project in. A solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

1. Start Visual Studio Code.
2. Select **File** > **Open Folder** from the main menu

A screenshot of a computer

Description automatically generated

1. In the **Open Folder** dialog, create a folder with name QuickKart and click **Select Folder**.
2. Open the **Terminal** in Visual Studio Code by selecting **…**> **Terminal** from the main menu. A screenshot of a computer

   Description automatically generated

The **Terminal** opens with the command prompt in the *QuickKart* folder.

A screenshot of a computer

Description automatically generated

1. In the **Terminal**, enter the following command:

> dotnet new sln

The terminal output looks like the following example:

A screenshot of a computer

Description automatically generated

## Create a class library project

Add a new .NET class library project named "QuickKartBL" to the solution.

1. In the terminal, run the following command to create the library project:

dotnet new classlib -o QuickKartBL

The -o or --output command specifies the location to place the generated output.

The terminal output looks like the following example:

A screenshot of a computer

Description automatically generated

1. Run the following command to add the library project to the solution:

dotnet sln add QuickKartBL/QuickKartBL.csproj

The terminal output looks like the following example:

A close up of a text

Description automatically generated

1. Check to make sure that the library targets .NET 6.

In **Explorer**, open QuickKartBL/QuickKartBL.csproj.

A screenshot of a computer

Description automatically generated

The TargetFramework element shows that the project targets .NET 8.0. If .NET 6.0 is your target, then change it accordingly.

A screenshot of a computer

Description automatically generated

1. Right click on Class1.cs. Rename it as StringLibrary. Open StringLibrary*.cs* and replace the code with the following code.

namespace QuickKartBL;

public static class StringLibrary

{

public static bool StartsWithUpper(this string? str)

{

if (string.IsNullOrWhiteSpace(str))

return false;

char ch = str[0];

return char.IsUpper(ch);

}

}

The class library, QuickKartBL.StringLibrary, contains a method named StartsWithUpper. This method returns a Boolean value that indicates whether the current string instance begins with an uppercase character. The Unicode standard distinguishes uppercase characters from lowercase characters. The Char.IsUpper(Char) method returns true if a character is uppercase.

StartsWithUpper is implemented as an extension method so that you can call it as if it were a member of the String class.

1. Save the file.
2. Run the following command to build the solution and verify that the project compiles without error.

dotnet build

**A screenshot of a computer program

Description automatically generated**

## Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. In the terminal, run the following command to create the console app project:

dotnet new console -o QuickKartTestApp

The terminal output looks like the following example:

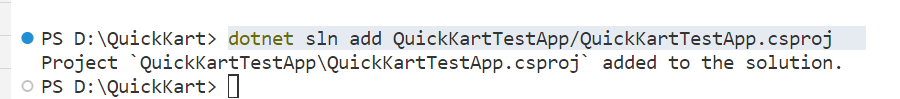
A screenshot of a computer

Description automatically generated

1. Run the following command to add the console app project to the solution:

dotnet sln add QuickKartTestApp/QuickKartTestApp.csproj

The terminal output looks like the following example:



1. Open *QuickKartTestApp/Program.cs* and replace all of the code with the following code.

using QuickKartBL;

class Program

{

static void Main(string[] args)

{

int row = 0;

do

{

if (row == 0 || row >= 25)

ResetConsole();

string? input = Console.ReadLine();

if (string.IsNullOrEmpty(input)) break;

Console.WriteLine($"Input: {input}");

Console.WriteLine("Begins with uppercase? " +

$"{(input.StartsWithUpper() ? "Yes" : "No")}");

Console.WriteLine();

row += 4;

} while (true);

return;

// Declare a ResetConsole local method

void ResetConsole()

{

if (row > 0)

{

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

Console.Clear();

Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a string and press <Enter>:{Environment.NewLine}");

row = 3;

}

}

}

The code uses the row variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the Enter key without entering a string, the application ends, and the console window closes.

1. Save your changes.

## Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. Run the following command:

dotnet add QuickKartTestApp/QuickKartTestApp.csproj reference QuickKartBL/QuickKartBL.csproj

The terminal output looks like the following example:

A screenshot of a computer

Description automatically generated

## Run the app

1. Run the following command in the terminal:

dotnet run --project QuickKartTestApp/QuickKartTestApp.csproj

1. Try out the program by entering strings and pressing Enter, then press Enter to exit.

The terminal output looks like the following example:

A screenshot of a computer

Description automatically generated

# Step-by-Step Guide to Using Microsoft SQL Server and Writing Queries

using Visual Studio Code

## Prerequisites

* Visual Studio Code with SQL Server (mssql) extension pack installed.

A screenshot of a computer

Description automatically generated

* Basic Understanding of writing DDL and DML queries in MS SQL Server

## SQL Server Instance Setup

Here we using the SQL Server Instance name as MSSQLLocalDB

### To Create an SQL Server Instance

1. Open the Terminal in VS Code and run the following command to check if Microsoft SQL Server is installed on the local device

sqllocaldb v

A close-up of a computer code

Description automatically generated

1. Run the following command to check the list of SQL Server Instances present in your local device.

sqllocaldb i

A close up of text

Description automatically generated

In case you do not have an instance created, you can create one with the following command

sqllocaldb create MSSQLLocalDB

A computer code with black text

Description automatically generated

[N.B.- You can also append the command with an optional parameter “-s” to start the instance as soon as created.]

### To Delete an SQL Server Instance

1. Run the following command to print the information about the specified LocalDB Instance.

sqllocaldb i MSSQLLocalDB

A screenshot of a computer program

Description automatically generated

1. Check the information. If the State is Running, the LocalDB Instance needs to be stopped before deleting it. Run the following command to stop a running LocalDB Instance.

sqllocaldb stop MSSQLLocalDB

A close-up of a computer screen

Description automatically generated

1. Run the following command to delete a stopped LocalDB Instance.

sqllocaldb delete MSSQLLocalDB

A close-up of a computer screen

Description automatically generated

## VSCode SQL Server (mssql) Extension Setup

### To Open SQL Server Object Explorer

You can perform either of the following steps to open SQL Server Object Explorer

1. Press ctrl + alt + D to open SQL Server Object Explorer
2. Press the  icon in the Activity Bar (Left Side, by default). If the icon is not present in the Activity Bar right-click on the bar and make sure SQL Server is selected

A screenshot of a computer

Description automatically generated

### To Add a SQL Server Instance to the Object Explorer

We will be working with the created instance named [MSSQLLocalDB](#_To_Create_an).

1. Open the Terminal and run the following command to check the information about the Instance named MSSQLocalDB

sqllocaldb i MSSQLLocalDB

A screenshot of a computer

Description automatically generated

1. Observe the State of the Instance. An Instance cannot be connected to if it is stopped and thus the Instance pipe name is empty. To start the Instance named MSSQLLocalDB run the following command

sqllocaldb start MSSQLLocalDB

A close-up of a computer code

Description automatically generated

1. Open the SQL Server Object Explorer and click on the Add Connection under Connections.

A screenshot of a computer

Description automatically generated

1. Write the connection string of the SQL Server Instance. For LocalDB Instance named MSSQLLocalDB the Connection String will be the following

Data Source=(localdb)\MSSQLLocalDB;

A screen shot of a computer

Description automatically generated

1. Write the Profile name for the connection profile and press Enter. The Instance name is considered as the profile name.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### To Write Queries and Execute it.

You can perform either step to start writing a query.

1. Create a new SQL file in a specified folder from the Explorer and start writing your query script.

A close-up of a computer screen

Description automatically generated

1. Right click on the Profile name in the SQL Server Object Explorer and select “New Query” and start writing your queries in a temporarily unsaved sql

A screenshot of a computer

Description automatically generated

After Opening the Script or the temporarily unsaved file you need to connect the Query Executor with your database using the Connection Profile if it is not connected. You can perform either step to connect the Query Executor with your database using the Connection Profile

1. In the Status Bar (Bottom Horizontal Bar) click on Disconnected to connect to a database

A screen shot of a computer

Description automatically generated

1. After writing your query click on the A green triangle in a white background

   Description automatically generated Execute button in the Editor Tabs. If not connected, it will ask you to connect to a database and automatically execute the query after the connection is established

A screenshot of a computer

Description automatically generated

[N.B – You can also trigger Execute Command by pressing Ctrl + Shift + E]

Select the MSSQLLocalDB Connection Profile. It will connect to the master database by default.

A screenshot of a computer

Description automatically generated

[N.B – In case you need to switch to a different database you can click on it again and select your database]

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

After the query execution, a new tab will open (right side, by default) to show the results of your query along with the message returned with proper timestamps and execution time.

# Step by step guide for creating a Data Access Layer using Database First Approach

## Prerequisites

* Visual Studio Code with the C# extension installed.
* If you have the C# Dev Kit extension installed, uninstall or disable it.
* The [.NET 6 SDK](https://dotnet.microsoft.com/download/dotnet/8.0) or NET 8 SDK
* Pre-built Database on Microsoft SQL Server or Azure SQL Server
* Connection String of the Database

Database First Approach is used when you already have a preexisting database hosted either locally or in a cloud environment. The connection string is required to scaffold the database ensuring proper Object Relation Mapping. In this guide, you will be able to create a Data Access Layer with Entity Framework Core by creating a repository of your database context.

## To Create a solution

Start by creating a blank solution to put the class library project in. A solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

1. Start Visual Studio Code.
2. Select **File** > **Open Folder** from the main menu

A screenshot of a computer

Description automatically generated

1. In the **Open Folder** dialog, create a folder with name QuickKart and click **Select Folder**.
2. Open the **Terminal** in Visual Studio Code by selecting **…**> **Terminal** from the main menu.

A screenshot of a computer

Description automatically generated

The **Terminal** opens with the command prompt in the *QuickKart* folder.

A close up of a screen

Description automatically generated

1. In the **Terminal**, enter the following command:

> dotnet new sln

The terminal output looks like the following example:

A screen shot of a computer

Description automatically generated

## To Create a class library project

Add a new .NET class library project named "QuickKart.DataAccessLayer" to the solution.

1. In the terminal, run the following command to create the library project:

dotnet new classlib -o QuickKart.DataAccessLayer

The -o or --output command specifies the location to place the generated output.

The terminal output looks like the following example:

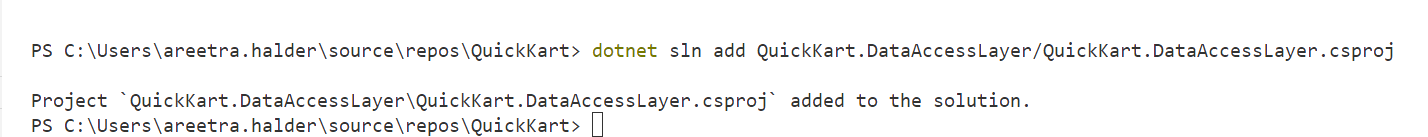
A screenshot of a computer

Description automatically generated

1. Run the following command to add the library project to the solution:

dotnet sln add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj

The terminal output looks like the following example:



1. Check to make sure that the library targets .NET 6.

In **Explorer**, open QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj

A screenshot of a computer

Description automatically generated

The TargetFramework element shows that the project targets .NET 8.0.

A screenshot of a computer program

Description automatically generated

1. Set the Nullable Property to disable.

A screenshot of a computer program

Description automatically generated

1. Run the following commands to download the required packages namely,
   * + Microsoft.EntityFrameworkCore

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore -v 8.0.0

* + - Microsoft.EntityFrameworkCore.SqlServer

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore.SqlServer -v 8.0.0

* + - Microsoft.EntityFrameworkCore.Tools

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore.Tools -v 8.0.0

* + - Microsoft.Extensions.Configuration

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration -v 8.0.0

* + - Microsoft.Extensions.Configuration.FileExtensions

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration.FileExtensions -v 8.0.0

* + - Microsoft.Extensions.Configuration.Json

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration.Json -v 8.0.0

This is how your solution file would look like after installing the six packages.

A screenshot of a computer program

Description automatically generated

To remove a package, you can use the dotnet remove command. Syntax: dotnet remove [PROJECT\_PATH] package [PACKAGE\_NAME]

Example: dotnet remove QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore

N.B – Specifying the version type is not required while removing the package. Dotnet CLI automatically takes care of it.

1. To use the Scaffold Command, install dotnet-ef tool by running the following command

dotnet tool install --global dotnet-ef –version 8.0

A screenshot of a computer

Description automatically generated

1. Write the following command to confirm the installation

dotnet ef

A screenshot of a computer

Description automatically generated

1. After installing dotnet-ef tool you can proceed with the following command. N.B – You would require the Connection String of your pre-built database to run the following command.

dotnet ef dbcontext scaffold "your-connection-string" Microsoft.EntityFrameworkCore.SqlServer -o Models --project QuickKart.DataAccessLayer\QuickKart.DataAccessLayer.csproj

A screenshot of a computer

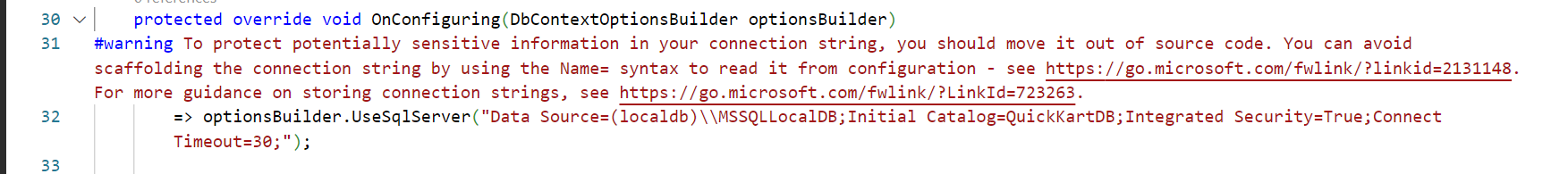
Description automatically generatedA computer screen shot of a computer code

Description automatically generated with medium confidence

Each Table of the Database has been Mapped into a folder named Models and a Database Context class has been generated in the folder namely QuickKartDbContext.cs

## Securing the Connection String

* 1. Open QuickKartDbContext.cs and observe OnConfiguring() Method of the class.



OnConfiguring Method is responsible for connecting to the database with the help of the connection string. Connection String is considered as a secret key. It is always a good practice to store the credentials at a separate place like Azure Key Vault or appsettings.json

* 1. Modify the OnConfiguring Method into a block method and write the below code.

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        var builder = new ConfigurationBuilder()

                          .SetBasePath(Directory.GetCurrentDirectory())

                          .AddJsonFile("appsettings.json");

      var config = builder.Build();

        var connectionString = config.GetConnectionString("QuickKartDBConnectionString");

        if (!optionsBuilder.IsConfigured)

        {

            optionsBuilder.UseSqlServer(connectionString);

        }

    }

Here, the class ConfigurationBuilder is used to build key/value based configuration settings for use in an application.

Directory.GetCurrentDirectory() - Gets the current working directory of the application

SetBasePath() - Sets the physical path of the file from the identified current working directory

AddJsonFile() - Provides the relative path to the base path

* 1. Create a New File named “appsettings.json” inside QuickKart.DataAccessLayer folder

A screenshot of a computer

Description automatically generated

* 1. Fill the following data in the JSON file

{

    "ConnectionStrings": {

        "QuickKartDBConnectionString": "YOUR\_CONNECTION\_STRING"

    }

}

The Database-First Approach set up using Entity Framework Core has been completed. To access the data in a centralized manner, it is recommended to create a repository class inside the project.

## To Create a Repository Class in the Class Library

.

1. Create a new file named QuickKartRepository.cs in QuickKartDataAccessLayer folder.

A screenshot of a computer

Description automatically generated

1. Write the following code in the newly created file

using QuickKart.DataAccessLayer.Models;

public class QuickKartRepository

{

    private readonly QuickKartDbContext \_dbContext;

    public QuickKartRepository(QuickKartDbContext dbContext)

    {

        this.\_dbContext = dbContext;

    }

    public List<Product> GetProducts(){

        List<Product> products;

        try

        {

            products = this.\_dbContext.Products.ToList();

        }

        catch (Exception)

        {

            products = null;

        }

        return products;

    }

}

The sample repository has been created. The class library project can now be referred to any other project for use. It is recommended to test the Data Access Layer in a separate Console Project.

## To Create a Test Application for DAL

Add a new .NET console application project named "QuickKart.TestApplication " to the solution.

1. In the terminal, run the following command to create a .Net Console Application Project in the QuickKart Directory

dotnet new console -o "QuickKart.TestApplication" --use-program-main

A white background with many small colored lines

Description automatically generated with medium confidence

Parameter “-o” or “-output” defines the output directory of the project generated and “—use-program-main” does not use top-level statements

1. Run the following command to add the library project to the solution:

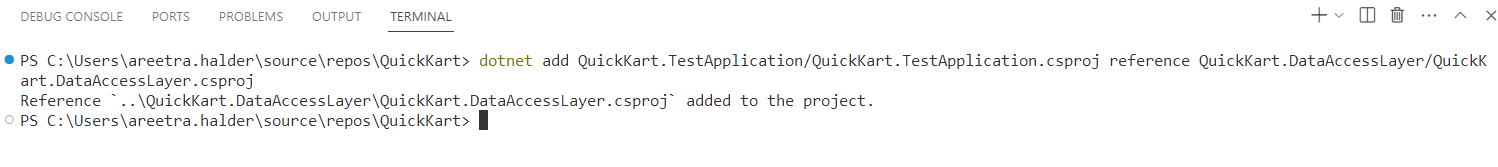
dotnet sln add QuickKart.TestApplication/QuickKart.TestApplication.csproj

A close-up of a computer screen

Description automatically generated

1. Run the following command to allow the Test Application refer to the Data Access Layer.

dotnet add QuickKart.TestApplication/QuickKart.TestApplication.csproj reference QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj



1. Write the following code in Program.cs to test the Data Access Layer

using QuickKart.DataAccessLayer;

using QuickKart.DataAccessLayer.Models;

namespace QuickKart.TestApplication;

class Program

{

    private static QuickKartRepository \_repository;

    static Program()

    {

        QuickKartDbContext dbContext = new QuickKartDbContext();

        \_repository = new QuickKartRepository(dbContext);

    }

    static void Main(string[] args)

    {

        TestGetProducts();

    }

    public static void TestGetProducts()

    {

        List<Product> products;

        Console.WriteLine("Testing GetProducts...");

        try

        {

            products = \_repository.GetProducts().Take(5).ToList();

            Console.WriteLine("{0,-20}{1,-30}{2,-20}{3,-20}{4,-20}", "Product Id",

            "Product Name",

            "Category Id",

            "Price",

            "Quantity Available"

            );

            foreach (var eachProduct in products)

            {

                Console.WriteLine("{0,-20}{1,-30}{2,-20}{3,-20}{4,-20}",

                    eachProduct.ProductId,

                    eachProduct.ProductName,

                    eachProduct.CategoryId,

                    eachProduct.Price,

                    eachProduct.QuantityAvailable

                );

            }

        }

        catch (Exception ex)

        {

            Console.WriteLine("Some Error Occured. Reason: " + ex.Message);

        }

        finally{

            System.Console.WriteLine("GetProducts Method has been executed.");

        }

    }

}

1. In the terminal move to QuickKart.DataAccessLayer with the following command.

cd QuickKart.DataAccessLayer

A close up of a computer screen

Description automatically generated

1. Run the following command in the Terminal to execute the Console Application Project and test the Data Access Layer.

dotnet run --project ../QuickKart.TestApplication/QuickKart.TestApplication.csproj

The terminal output looks like the following example.

A screenshot of a computer

Description automatically generated

The Data Access Layer securely connects to the SQL Server Instance with the help of the Connection String from appsettings.json file present in the Current Working Directory. Thus, it works only when the working directory is inside QuickKart.DataAccessLayer. In case, the working directory is changed make sure appsettings.json is present in the updated path.

# Step by step guide for creating a Data Access Layer using Code First Approach

## Prerequisites

* Visual Studio Code with the C# extension installed.
* If you have the C# Dev Kit extension installed, uninstall or disable it.
* The [.NET 6 SDK](https://dotnet.microsoft.com/download/dotnet/8.0) or NET 8 SDK
* Connection String of the Database

## Code First Approach is employed when initiating development without a preexisting database. The connection string is essential for generating the database schema based on the defined code, facilitating Object-Relational Mapping. This guide enables the creation of a Data Access Layer using Entity Framework Core by establishing a repository for the database context.

## To Create a solution

Start by creating a blank solution to put the class library project in. A solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

1. Start Visual Studio Code.
2. Select **File** > **Open Folder** from the main menu

A screenshot of a computer

Description automatically generated

1. In the **Open Folder** dialog, create a folder with name QuickKart and click **Select Folder**.
2. Open the **Terminal** in Visual Studio Code by selecting **…**> **Terminal** from the main menu.

A screenshot of a computer

Description automatically generated

The **Terminal** opens with the command prompt in the *QuickKart* folder.

A close up of a screen

Description automatically generated

1. In the **Terminal**, enter the following command:

> dotnet new sln

The terminal output looks like the following example:

A screen shot of a computer

Description automatically generated

## To Create a class library project

Add a new .NET class library project named "QuickKart.DataAccessLayer" to the solution.

1. In the terminal, run the following command to create the library project:

dotnet new classlib -o QuickKart.DataAccessLayer

The -o or --output command specifies the location to place the generated output.

The terminal output looks like the following example:

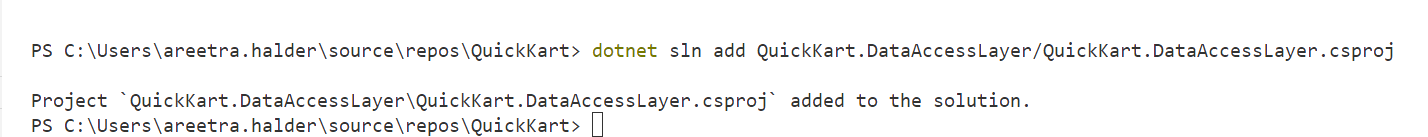
A screenshot of a computer

Description automatically generated

1. Run the following command to add the library project to the solution:

dotnet sln add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj

The terminal output looks like the following example:



1. Check to make sure that the library targets .NET 6.

In **Explorer**, open QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj

A screenshot of a computer

Description automatically generated

The TargetFramework element shows that the project targets .NET 8.0.

A screenshot of a computer program

Description automatically generated

1. Set the Nullable Property to disable.

A screenshot of a computer program

Description automatically generated

1. Run the following commands to download the required packages namely,
   * + Microsoft.EntityFrameworkCore

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore -v 8.0.0

* + - Microsoft.EntityFrameworkCore.SqlServer

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore.SqlServer -v 8.0.0

* + - Microsoft.EntityFrameworkCore.Tools

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore.Tools -v 8.0.0

* + - Microsoft.Extensions.Configuration

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration -v 8.0.0

* + - Microsoft.Extensions.Configuration.FileExtensions

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration.FileExtensions -v 8.0.0

* + - Microsoft.Extensions.Configuration.Json

dotnet add QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.Extensions.Configuration.Json -v 8.0.0

This is how your solution file would look like after installing the six packages.

A screenshot of a computer program

Description automatically generated

To remove a package, you can use the dotnet remove command. Syntax: dotnet remove [PROJECT\_PATH] package [PACKAGE\_NAME]

Example: dotnet remove QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj package Microsoft.EntityFrameworkCore

N.B – Specifying the version type is not required while removing the package. Dotnet CLI automatically takes care of it.

1. To apply Code Migration for Database, install dotnet-ef tool by running the following command

dotnet tool install --global dotnet-ef –version 8.0

A screenshot of a computer

Description automatically generated

1. Write the following command to confirm the installation

dotnet ef

A screenshot of a computer

Description automatically generated

1. In the terminal move to QuickKart.DataAccessLayer with the following command.

cd QuickKart.DataAccessLayer

A close up of a computer screen

Description automatically generated

1. Create a file named Product.cs in the Models folder and write the following code.

using System;

using System.Collections.Generic;

namespace QuickKart.DataAccessLayer.Models;

public partial class Product

{

    public string ProductId { get; set; }

    public string ProductName { get; set; }

    public decimal Price { get; set; }

    public int CategoryId { get; set; }

    public int QuantityAvailable { get; set; }

    public Category Category { get; set; }

}

1. Create a file named Category.cs in the Models folder and write the following code.

using System;

using System.Collections.Generic;

namespace QuickKart.DataAccessLayer.Models;

public partial class Category

{

    public Category()

    {

        Products = new HashSet<Product>();

    }

    public int CategoryId { get; set; }

    public string CategoryName { get; set; }

    public ICollection<Product> Products { get; set; }

}

1. Create a file named QuickKartDbContext.cs in the Models folder and write the following code.

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

namespace QuickKart.DataAccessLayer.Models;

public partial class QuickKartDbContext : DbContext

{

    public virtual DbSet<Product> Products { get; set; }

    public virtual DbSet<Category> Categories { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        var builder = new ConfigurationBuilder()

                          .SetBasePath(Directory.GetCurrentDirectory())

                          .AddJsonFile("appsettings.json");

        var config = builder.Build();

        var connectionString = config.GetConnectionString("QuickKartDBConnectionString");

        if (!optionsBuilder.IsConfigured)

        {

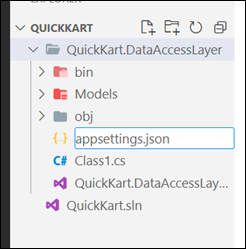
            optionsBuilder.UseSqlServer(connectionString);

        }

    }

}

1. Create a New File named “appsettings.json” inside QuickKart.DataAccessLayer folder



1. Fill the following data in the JSON file

{

    "ConnectionStrings": {

        "QuickKartDBConnectionString": "YOUR\_CONNECTION\_STRING"

    }

}

1. You can now proceed with the following command to add a migration by a given name. It is recommended to ensure proper naming of your migrations.

Syntax:

dotnet ef migrations add “MIGRATION\_NAME” [-o “OUTPUT\_PATH”]

Optional Parameter -o Ensures the output is present in a mentioned path. By Default, its value is “Migrations”

Usage:

dotnet ef migrations add InitialMigration

N.B – The output will be generated in the Migrations folder since the default value of -o is “Migrations”

A screenshot of a computer

Description automatically generated A close-up of a text

Description automatically generated

1. To Update the Database with the added Migrations Run the following command

dotnet ef database update

A screenshot of a computer code

Description automatically generated

The Code First Approach set up using Entity Framework Core has been completed. To access the data in a centralized manner, it is recommended to create a repository class inside the project.

## To Create a Repository Class in the Class Library

.

1. Create a new file named QuickKartRepository.cs in QuickKartDataAccessLayer folder.

A screenshot of a computer

Description automatically generated

1. Write the following code in the newly created file

using QuickKart.DataAccessLayer.Models;

namespace QuickKart.DataAccessLayer;

public class QuickKartRepository

{

    private readonly QuickKartDbContext \_dbContext;

    public QuickKartRepository(QuickKartDbContext dbContext)

    {

        this.\_dbContext = dbContext;

    }

    public String AddCategory(Category category){

        String result = "Failed";

        try

        {

            \_dbContext.Categories.Add(category);

            int rowsAffected = \_dbContext.SaveChanges();

            if(rowsAffected > 0){

                result = "Success";

            }

        }

        catch (Exception ex)

        {

            result = String.Format("Failed. Reason: {0}", ex.Message);

        }

        return result;

    }

}

The sample repository has been created. The class library project can now be referred to any other project for use. It is recommended to test the Data Access Layer in a separate Console Project.

## To Create a Test Application for DAL

Add a new .NET console application project named "QuickKart.TestApplication " to the solution.

1. In the terminal, run the following command to create a .Net Console Application Project in the QuickKart Directory

dotnet new console -o "QuickKart.TestApplication" --use-program-main

A white background with many small colored lines

Description automatically generated with medium confidence

Parameter “-o” or “-output” defines the output directory of the project generated and “—use-program-main” does not use top-level statements

1. Run the following command to add the library project to the solution:

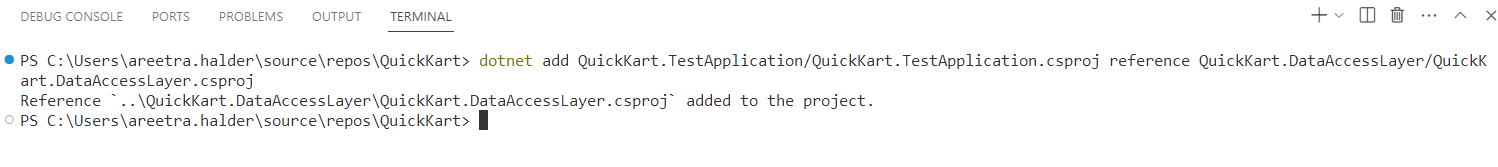
dotnet sln add QuickKart.TestApplication/QuickKart.TestApplication.csproj

A close-up of a computer screen

Description automatically generated

1. Run the following command to allow the Test Application refer to the Data Access Layer.

dotnet add QuickKart.TestApplication/QuickKart.TestApplication.csproj reference QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj



1. Write the following code in Program.cs to test the Data Access Layer

using QuickKart.DataAccessLayer.Models;

namespace QuickKart.DataAccessLayer;

public class QuickKartRepository

{

    private readonly QuickKartDbContext \_dbContext;

    public QuickKartRepository(QuickKartDbContext dbContext)

    {

        this.\_dbContext = dbContext;

    }

    public String AddCategory(Category category){

        String result = "Failed";

        try

        {

            \_dbContext.Categories.Add(category);

            int rowsAffected = \_dbContext.SaveChanges();

            if(rowsAffected > 0){

                result = "Success";

            }

        }

        catch (Exception ex)

        {

            result = String.Format("Failed. Reason: {0}", ex.Message + ex.InnerException.Message);

        }

        return result;

    }

}

1. In the terminal move to QuickKart.DataAccessLayer with the following command.

cd QuickKart.DataAccessLayer

A close up of a computer screen

Description automatically generated

1. Run the following command in the Terminal to execute the Console Application Project and test the Data Access Layer.

dotnet run --project ../QuickKart.TestApplication/QuickKart.TestApplication.csproj

The terminal output looks like the following example.

A close-up of a computer code

Description automatically generated

The Data Access Layer securely connects to the SQL Server Instance with the help of the Connection String from appsettings.json file present in the Current Working Directory. Thus, it works only when the working directory is inside QuickKart.DataAccessLayer. In case, the working directory is changed make sure appsettings.json is present in the updated path.

# Step by Step Guide for creating a Service Layer in ASP .NET Web API

## Prerequisites

* Visual Studio Code with the C# extension installed

A screenshot of a computer

Description automatically generated

* If you have the C# Dev Kit extension installed, uninstall, or disable it.
* The [.NET 6 SDK](https://dotnet.microsoft.com/download/dotnet/8.0) or NET 8 SDK
* Pre-built Database on Microsoft SQL Server or Azure SQL Server
* Connection String of the Database
* Pre-Existing Data Access Layer
* [Optional] Material Icon Theme Extension installed and enabled.

A screenshot of a computer

Description automatically generated

## To create an ASP .NET Web API Project

We will be continuing from the completion of QuickKart.DataAccessLayer Project of QuickKart.sln here.



1. Open the Terminal and run the following command in the QuickKart Directory to create an ASP .NET Web API Project

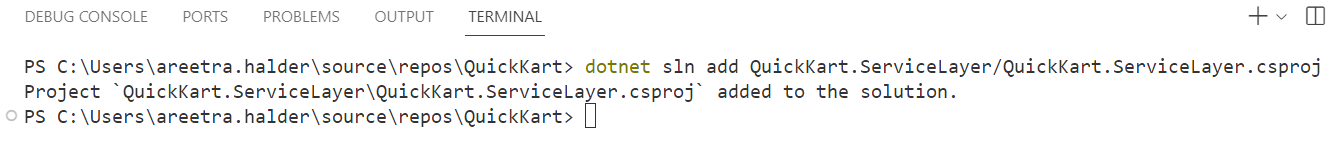
dotnet new webapi --use-controllers --use-program-main -o QuickKart.ServiceLayer

A screenshot of a computer code

Description automatically generated

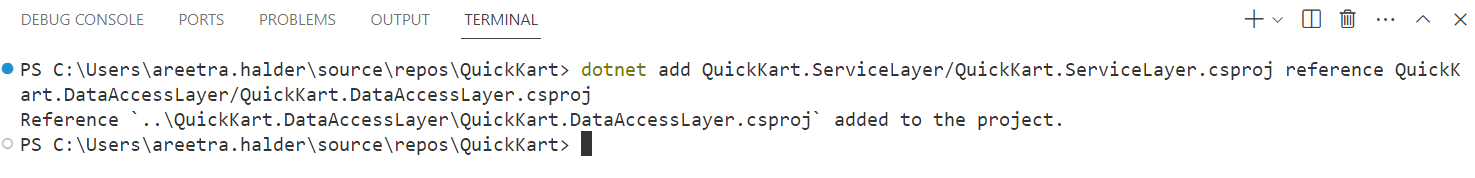
1. Run the following command to add the ASP.NET Web API Project to the solution file named QuickKart.sln

dotnet sln add QuickKart.ServiceLayer/QuickKart.ServiceLayer.csproj



1. Run the following command to add the project reference of the Data Access Layer into Service Layer

dotnet add QuickKart.ServiceLayer/QuickKart.ServiceLayer.csproj reference QuickKart.DataAccessLayer/QuickKart.DataAccessLayer.csproj



1. Open QuickKart.ServiceLayer.csproj file in QuickKart.ServiceLayer Directory and Set Nullable to disable and InvariantGlobalization to false

A screenshot of a computer program

Description automatically generated

In case you are having different settings or configurations for development and deployment (for example, using database for testing purpose) it is recommended to use appsettings.Development.json and appsettings.json respectively. You can also add a gitignore file and add appsettings.Development.json file there. However, we will be only focusing on appsettings.json as of now.

1. Modify appsettings.json and add the Connection String of QuickKartDB there. The appsettings.json file would be looking like this after modification.

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "AllowedHosts": "\*",

  "ConnectionStrings": {

    "QuickKartDBConnectionString": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=QuickKartDB;Integrated Security=True;"

  }

}

1. [Optional] You can remove the following files in QuickKart.ServiceLayer Directory as these are sample files only
   * Controllers/WeatherForecastController.cs
   * WeatherForecast.cs
2. Create a new Folder named utils to store the utilities required for the project
3. Create a new file named ApiResponse.cs and paste the following code



During Production, Apart from Error Handling, Error Logging becomes necessary for developers to know what the error is in a proper format. Thus, ApiResponse is created as a Generic Class which is used here to store the response of the API in a centralized manner. The same has been implemented in the Data Access Layer named RepositoryResponse.cs to ensure cleanliness.

1. Create a New File named ProductController.cs in the Controllers Folder and write the following code



1. To add Dependency Injection in the Controller we need to use Add Transient Method in Program.cs. Modify the following code in Program.cs file from Line 19 and paste the following code

        builder.Services.AddTransient<QuickKartDbContext>();

        builder.Services.AddTransient<QuickKartRepository>(

            c => new QuickKartRepository(c.GetRequiredService<QuickKartDbContext>()));

1. Open the Terminal and run the following command in the QuickKart.ServiceLayer folder

dotnet run

A screenshot of a computer program

Description automatically generated

To run the dotnet project in watch mode execute the following command in the Terminal

dotnet watch run

A screenshot of a computer

Description automatically generated